X-Force: Force-Executing Binary Programs for Security Applications

Fei Peng, Zhui Deng, Xiangyu Zhang, Dongyan Xu
Purdue University
Zhiqiang Lin
UT Dallas
Zhendong Su
UC Davis

Outline

- Background & Motivation
- Design
- Implementation Challenges
- Evaluation
- Conclusion

Background & Motivation

- Binary analysis
  - The analysis on compiled binary software
    - No source code & symbolic information
  - More challenging than software analysis using source code
    - Control flow graph, variable type

- Binary analysis has many security applications
  - Exposing malware behavior by constructing CFG/CG
  - Identifying and patching security vulnerabilities

Background & Motivation

- Existing approaches
  - Static analysis (IDA)
    - Examining the code without executing it
  - Dynamic analysis (Valgrind, PIN)
    - Testing and evaluation of an application during runtime
  - Symbolic analysis (BitBlaze, S2E)
    - Determine what inputs cause each part of the program to execute

	Good Coverage	Packing & Obfuscation	Precision	Scalability
Static	✓	✗	✗	✓
Dynamic	✗	✓	✓	✓
Symbolic	✓	✗	✓	✗

Outline

- Background & Motivation
- Design
- Implementation Challenges
- Evaluation
- Conclusion

Design

- What is X-Force?
  - Dynamic analysis engine that forces a binary to execute
    - Provide no inputs or any environment setup
    - Explore different paths by simply switching the outcome of predicates

Example -Hijack the name resolution for a specific domain

```
DNSentry *p;
void main () {
        int x = inputInt ();
        if (C (x))
                p = (DNSentry *)malloc (...);
        if (x & CODE_RED) {
                genName (x, p);
                hashTablePut (x, p);
        }
        ...
        hashTablePut (...., o); // o is of type T
        ...
        s = hashTableGet (y); // y == x through execution
        if (s)
                //redirection for the domain specified by s
                redirection ();
}

void genName (int x, DNSentry *q) {
        inputDirectionary ();
        *(q->name) = ...Lookup (x, date ())...;
}
```

Reads an integer x

If condition satisfied, a DNS object is allocated to p

If CODE_RED bit set is in x, get the domain name

Fetch an object using key y==x

Other objects are put into the hash table

If an object is fetched successfully, malicious payload triggered

Example – Static Analysis

```
1  DNSentry *p;
2  void main () {
3          int x = inputInt ();
4          if (C (x))
5                  p = (DNSentry *)malloc (...);
6          if (x & CODE_RED) {
7                  genName (x, p);
8                  hashTablePut (x, p);
9          }
10         ...
11         hashTablePut (..., o); // o is of type T
12         ...
13         s = hashTableGet (y); // y == x through execution
14         if (s)
15                 //redirection for the domain specified by s
16                 redirection ();
17 }
18
19 void genName (int x, DNSentry *q) {
20         inputDirectionary ();
21         *(q->name) = ...Lookup (x, date ())...;
22 }
```

Object fetched at 13 is from either 8 or 11

Truth: Only from 8

Example – Dynamic Analysis

```
1  DNSentry *p;
2  void main () {
3          int x = inputInt ();
4          if (C (x))
5                  p = (DNSentry *)malloc (...);
6          if (x & CODE_RED) {
7                  genName (x, p);
8                  hashTablePut (x, p);
9          }
10         ...
11         hashTablePut (..., o); // o is of type T
12         ...
13         s = hashTableGet (y); // y == x through execution
14         if (s)
15                 //redirection for the domain specified by s
16                 redirection ();
17 }
18
19 void genName (int x, DNSentry *q) {
20         inputDirectionary ();
21         *(q->name) = ...Lookup (x, date ())...;
22 }
```

If CODE_RED bit is not set in x, an object with key x is not put into hash table

Fetching object at line 13 fails

Malicious payload is not triggered

Example – Symbolic Analysis

```
DNSentry *p;
void main () {
        int x = inputInt ();
        if (C (x))
                p = (DNSentry *)malloc (...);
        if (x & CODE_RED) {
                genName (x, p);
                hashTablePut (x, p);
        }
        ...
        hashTablePut (..., o); // o is of type T
        ...
        s = hashTableGet (y); // y == x through execution
        if (s)
                //redirection for the domain specified by s
                redirection ();
}

void genName (int x, DNSentry *q) {
        inputDirectionary ();
        *(q->name) = ...Lookup (x, date ())...;
}
```

Model x as symbolic variable, hidden payload may be reached

File processing requires tremendous work due to nontrivial file size and format

Example – X-Force

```
DNSentry *p;
void main () {
        int x = inputInt ();
        if (C (x))
                p = (DNSentry *)malloc (...);
        if (x & CODE_RED) {
                genName (x, p);
                hashTablePut (x, p);
        }
        ...
        hashTablePut (..., o); // o is of type T
        ...
        s = hashTableGet (y); // y == x through execution
        if (s)
                //redirection for the domain specified by s
                redirection ();
}

void genName (int x, DNSentry *q) {
        inputDirectionary ();
        *(q->name) = ...Lookup (x, date ())...;
}
```

Provides random inputs

Assume all 3 predicates go to false branch

Leads to a non-interesting path

Example – X-Force

```
1 DNSentry *p;
2 void main () {
3         int x = inputInt ();
4         if (C (x))
5                 p = (DNSentry *)malloc (...);
6         if (x & CODE_RED) {
7                 genName (x, p);
8                 hashTablePut (x, p);
9         }
10        ...
11        hashTablePut (..., o); // o is of type T
12        ...
13        s = hashTableGet (y); // y == x through execution
14        if (s)
15                //redirection for the domain specified by s
16                redirection ();
17 }
18
19 void genName (int x, DNSentry *q) {
20        inputDirectionary ();
21        *(q->name) = ...Lookup (x, date ())...;
22 }
```

Flip predicates one by one

Flip predicate at line 4 first, line 5 gets covered

Leads to a non-interesting path

Example – X-Force

```
1  DNSentry *p;
2  void main () {
3          int x = inputInt ();
4          if (C (x))
5                  p = (DNSentry *)malloc (...);
6          if (x & CODE_RED) {
7                  genName (x, p);
8                  hashTablePut (x, p);
9          }
10         ...
11         hashTablePut (..., o); // o is of type T
12         ...
13         s = hashTableGet (y); // y == x through execution
14         if (s)
15                 //redirection for the domain specified by s
16                 redirection ();
17 }
18
19 void genName (int x, DNSentry *q) {
20         inputDirectionary ();
21         *(q->name) = ...Lookup (x, date ())...;
22 }
```

Not covered, p = NULL

Flip predicate at line 6

Memory write exception, crash!

Crash-free Execution

- Ideas on memory access exception
  - Skip it?
    - A lot of following exceptions, cascading effect on program state corruption
    - Lose heap data
  - Allocate a piece of memory on demand
    - It is not sufficient by just fixing the corrupted pointer itself
    - Fix the other correlated pointers

Example – Dataflow

```
1  DNSentry *p;
2  void main () {
3        int x = inputInt ();
4        if (C (x))
5              p = (DNSentry *)malloc (...);
6        if (x & CODE_RED) {
7              genName (x, p);
8              hashTablePut (x, p);
9        }
10       ...
11       hashTablePut (..., o); // o is of type T
12       ...
13       s = hashTableGet (y); // y == x through execution
14       if (s)
15             //redirection for the domain specified by s
16             redirection ();
17 }
18
19 void genName (int x, DNSentry *q) {
20       inputDirectionary ();
21       *(q->name) = ...Lookup (x, date ())...;
22 }
```

Not covered, p = NULL

Flip predicate at line 6

Memory write exception, crash!

Crash-free Execution

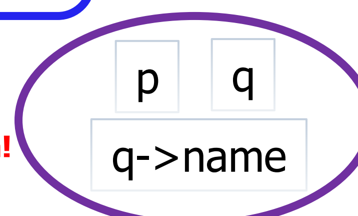
- Observations
  - Some pointers are correlated
  - Correlated pointers are only linearly correlated
    - No multiplication/division
- Solution – Linear set tracing
  1. Memories/registers that are linearly correlated are put into a set
     - Copying (e.g. b = a)
     - Adding or subtracting (e.g. q = p +/- 4)
  2. When memory exception occurs, recover values for elements based on maintained linear sets

Example – Linear Set Tracing

```
DNSentry *p;
void main () {
        int x = inputInt ();
        if (C (x))
                p = (DNSentry *)malloc (...);
        if (x & CODE_RED) {
                genName (x, p);
                hashTablePut (x, p);
        }
        ...
        hashTablePut (..., o); // o is of type T
        ...
        s = hashTableGet (y); // y == x through execution
        if (s)
                //redirection for the domain specified by s
                redirection ();
}

void genName (int x, DNSentry *q) {
        inputDirectionary ();
        *(q->name) = ...Lookup (x, date ())...;
}
```

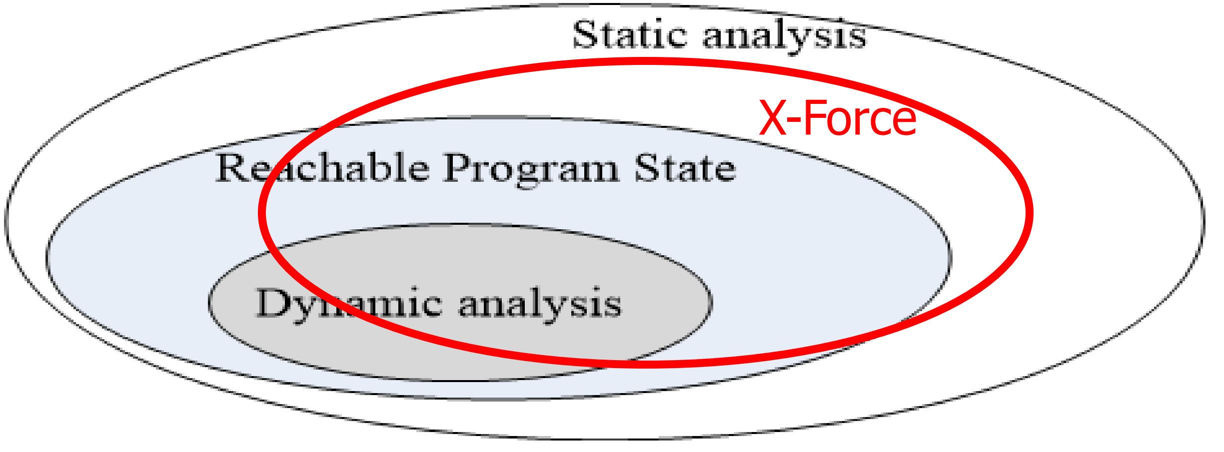
Memory write exception, crash!



Path Exploration

- Exploration algorithms
  - Branch coverage driven algorithm
    - Number of executions - O(n)
      - n denotes the number of basic blocks
  - Exponential search algorithm - $O(2^n)$
- Implement a taint analysis subsystem
  - Determine branches that are input related

The Essence of X-Force



- Reachable program state
  - Ideal coverage
- Static analysis
  - Over-approximate coverage
- Dynamic analysis
  - Under-approximate coverage
- X-Force
  - Practicality

The Essence of X-Force

- X-Force is important in practice
  - Results are not affected much by infeasible paths
    - Only a small number of predicates are switched
  - Fast
  - Naturally handle packed, obfuscated, and even self-modifying binaries
  - Existing dynamic analysis can be easily ported to X-Force

Outline

- Background & Motivation
- Design
- Implementation Challenges
- Evaluation
- Conclusion

Implementation Challenges

- Indirect jump - Jump Table
  - Leverage existing jump table reverse engineering techniques
  - Treat them as direct conditional branch in exploration algorithms
- Loops
  - If the loop bound is computed from input, it may be a corrupted value
    - Use taint analysis subsystem to determine if it's input related
    - If so, set the loop bound to a pre-defined constant
- Recursions
  - Maintain call stack during execution to detect recursion
  - If recursion is too deep, skip calling into it by simulating a return instruction

Implementation Challenges

- Handling library function calls
  - I/O functions, memory manipulation functions
- Protect stack memory
  - return addresses, base pointers
- Handling multiple thread execution
  - Serialize the execution
  - Explore different thread scheduling

Outline

- Background & Motivation
- Design
- Technical Challenges
- Evaluation
- Conclusion

Evaluation: Case Study I – CFG/CG Construction

- Instruction Coverage

	IDA	Dynamic	X-Force	Dynamic \ X-Force	X-Force \ Dynamic
164.gzip	7913	3601	5075	0	1474
175.vpr	31847	19409	29218	0	9820
176.gcc	310277	157451	227546	0	70095
181.mcf	2184	1622	1935	0	313
186.crafty	43327	27811	42763	0	14952
197.parser	25532	17339	23135	0	5796
252.eon	70592	15580	27224	0	11644
253.perlbmk	132264	55964	33643	28961	6640
254.gap	113410	37564	110066	0	72502
255.vortex	132053	53798	101207	0	47409
256.bzip2	5761	3612	4830	0	1218
300.twolf	46556	19996	41935	0	21939

Evaluation: Case Study I – CFG/CG Construction

- Indirect Call Edge Coverage

	IDA	Dynamic	LLVM	X-Force	Dynamic \ X-Force	X-Force \ Dynamic
164.gzip	0	2	2	2	0	0
176.gcc	25	169	9141	1720	0	1551
252.eon	0	60	28802	121	0	61
253.perlbmk	24	225	-	151	122	48
254.gap	2	1103	187155	20485	0	19382
255.vortex	0	28	340	30	0	2

Evaluation: Case Study I – CFG/CG Construction

- Performance

	Running Time (s)	# of Runs	Avg. Switched Predicates # / Total #
164.gzip	704	246	2.1/1291
175.vpr	8725	1849	4.7/2164
176.gcc	173241	26606	12.9/29847
181.mcf	129	113	4.3/153
186.crafty	43995	2496	8.0/62582
197.parser	3424	1820	6.4/944
252.eon	6379	2091	4.1/3146
253.perlbmk	7137	843	8.3/9535
254.gap	50745	7319	6.0/173316
255.vortex	34776	8566	7.3/2548
256.bzip2	557	209	1.4/7001
300.twolf	10043	2825	5.4/1322

Evaluation: Case Study II – Malware Analysis

Name	MD5	File Size(KB)	Number of Library Call Sites		
			IDA Pro	Native Run	X-Force
dg003.exe	4ec0027bef4d7e1786a04d021fa8a67f	192	808	546	1750
Win32/PWSteal.F	04eb2e58a145462334f849791bc75d18	20	9	28	94
APT1.DAIRY	995442f722cc037885335340fc297ea0	19	213	68	236
APT1.GREENCAT	0c5e9f564115bfcbee66377a829de55f	14.5	303	114	302
APT1.HELAUTO	47e7f92419eb4b98ff4124c3ca11b738	8.5	109	33	109
APT1.STARSYPOUND	1f2eb7b090018d975e6d9b40868c94ca	7	80	15	80
APT1.WARP	36cd49ad631e99125a3bb2786e405cea	45.5	495	156	414
APT1.NEWSREEL	2c49f47c98203b110799ab622265f4ef	21	189	49	192
APT1.GOGGLES	57f98d16ac439a11012860f88db21831	10.5	127	45	131
APT1.BOUNCER	6ebd05a02459d3b22a9d4a79b8626bf1	56	24	39	562

- X-Force discovers more lib calls than IDA for packed/obfuscated malware
- X-Force beats dynamic native run for all the programs

Evaluation: Case Study III – Type Reverse Engineering

- REWARDS
  - A dynamic analysis tool of type reverse engineering
- Porting REWARDS to X-Force
  - X-Force provides concrete execution states that are used by REWARDS
  - Little modification
- Results
  - Increase variable coverage from 57% to 84%
  - Increase type reverse accuracy from 88% to 90%

References

- Static analysis
  - Codesurfer/x86
  - IDA-Pro
  - Tie
- Dynamic analysis
  - Dart
  - REWARDS
  - Howard
  - Panorama
- Symbolic analysis
  - KLEE
  - S2E
  - BitBlaze

Outline

- Background & Motivation
- Design
- Technical Challenges
- Evaluation
- Conclusion

Conclusion

- Propose dynamic analysis engine X-Force, a system that can force binary to be executed
  - Requiring no inputs or any environment setup
- Develop a crash-free execution model
  - Detect and recover exceptions properly.
- Develop various execution path exploration algorithms
  - Provide customized options for users to reduce search spaces
- Evaluate X-Force on 3 types of case studies
  - CFG/CG construction
  - Malware analysis
  - Type reverse engineering

Thank you!

Q & A